Project Title: Len Den – Money Management

CATEGORY: Web & Mobile Application

Group Members:

Pradyumna Bhardwaj (1801124)

Rishabh Niranjan (1801140)

Rishabh Yadav (1801141)

ABSTRACT

With today's modernizing world the means of enjoyment have changed from in house parties to outdoor clubs, increasing the burden of splitting bills. The students with their high frequency of small transactions see their allowances dry up, people nowadays are more worried amount how'd they split the bill than are excited to have fun on a night out. Some people we interviewed had to say –

"When I go out with my friends, I want to track all the expenses without having to bother about splitting bills while we're out having fun."

"I don't want to forget tracking the multiple shared expenses I have with my roommates which might lead to confusion."

All this fear of splitting bill among friends is due to the clumsy process of who owes who, calculating what each person owes and then contacting each one of them about days old transactions. These tedious steps have been automated inside our LENDEN application to prevent our users from worrying about splitting bill and enjoy their outing to the fullest.

INTRODUCTION

It is a simple ledger application which helps in keeping account of the credit and debit history among friends. Users can keep track of the total money lent and owed. We have especially taken the needs of students in

mind who find it difficult to keep a record of their ‘udhaar khata’. Due to their numerous small transactions, they quickly find themselves in debt. Not able to effectively co-ordinate the money lent among a group of friends after an outing is also a major concern for students as they find following up and explaining the whole transaction history to everyone of the group individually tiresome. Students who are close friends or are roommates usually buy large number of necessary items for their room which are hard to keep track of as they think such small amounts of 30-50 rupees can be forgotten but these small transactions pile up to finally exhaust their allowances.

Many other ledger apps can be found on the internet today like OkCredit, split wise, etc. but all of them lack one or the other of the following-

- Chatting option with the other person
- Flexible in handling groups expenses
- Availability on both web and mobile platforms

We’ve added features in our application to tackle these problems as well as many more. For instance, to tackle the above problems –

- we’ve added a chatting feature into our app this helps reduce confusion and dispute regarding any bill then there on our platform. This feature has also been extended to groups which can be easily created on our application, this way we are hoping to reduce the monotonous job of explaining the bills individually to every person.
- Groups are easy to make and operate on our platform reducing the extra work of creating a separate group for bill matters on other social media platform and splitting the due amount among the group members.
- Our application is available both as a website and as a mobile based application increasing the convenience of the users

Some other problems we saw with the current day ledger applications were–

- unauthorized email could be added – we’ve added a feature for email verification & authorization.
- problem with deleting mistakenly added transaction – feature for deleting such transactions.

- complex UI – our UI was made keeping in mind how complex today's apps are and to completely satisfy our users.
- notifications – we have added a feature for e-mail notification
- problem with compiling complete history of money lent ad owed – feature to prevent such hassle.

Product Description

It helps students to record credit (Len) and debit (Den) transactions simply. it is a digital ledger money book or len den app that you just will simply maintain on your smartphone or browser. Gone are the times once people maintain daily hisab kitaab manually on udhar khata register. This digital ledger book would assist you simply maintain hisab kitaab of friends and family alike by permitting you to enter credit and debit records. Now there would be no worry about forgetting the transactions. Students often suffer losses due to inaccurate entries in ledgers or bad records.

This product aims to bring together all feasibility in accessing the transactions, giving flexibility and facilitating the ledger process. The product main focus is to be robust in operation and user friendly in usage.

1. Platforms are available: mobile application which is compatible with android operating system and the web application.

2. Sign in option:

    - username & password
    - email verification

User Characteristics

User - Client-side User

- Able to create and login to his/her account.
- Able to modify user profile.
- Able to maintain chats / transactions within the conversations he/she is part of.
- Able to add/delete conversation with his/her connections.
- Able to maintain connections with friends and family.

Assumptions

We assume:
1. All users have basic knowledge in English language.
2. All users have basic knowledge in computer and smartphone usage.
3. User must have internet connection while using the application.

Constraints

1. All users have to be logged in in order to use the product and to access the information

2. Android version of mobile phone of the users is another problem, few of the student are likely to have cheaper phones and that can be a problem for the application usage as the standard requirement of application is Android7.0+

Dependencies

As it is a both mobile & web-based application requiring data to be fetched from cloud database, it is always dependent on internet connection. A connection is required to send commands and receive answers, usually in the form of a result set.

REQUIREMENTS

The following definitions are intended as a guideline to prioritize requirements.

- Priority 1 – The requirement is a “must have” as outlined by policy/law
- Priority 2 – The requirement is needed for improved processing, and the fulfilment of the requirement will create immediate benefits
- Priority 3 – The requirement is a “nice to have” which may include new functionality

#	Functional Requirement	Description	Priority
1	Secured Authentication	Users must be authenticated to login and create their profiles.	1
2	Add Transactions	User can add Credit/Debit entry	1
3	Modify Transactions	User can edit and Delete their Transaction Entries within conversation.	1
4	Modify User Profiles	User can customize their profile. Also, they can edit the names of their contacts.	2
5	Ease of Access	Simple UI for easy navigations & Management.	1
6	Notify Connections	User can send mails and notify contacts about the due payment.	3
7	View Summary	User can get a whole summary of debit and credits in one place to reduce hassle.	1
8	Group Splitting	User can make a group of contact and split the bill among them and track the status.	1
9	Add Connections	User can add connections from maintaining expenses manually via user_id.	1

Functional Requirements

1. Secured Authentication: Our basic and most necessary functional requirement was to make user sign and sign secured and to make sure user is authenticated to perform all the possible actions.
   Our web and portable application are associated with same backend. Thus, security of the application is dealt with by the backend in the accompanying manners:
   - Ensures that clients and customer applications are recognized and that their characters are appropriately confirmed.
   - Ensures that clients and customer applications can just access information and administrations for which they have been appropriately approved.
   - Detects endeavoured interruptions by unapproved people and customer applications.

2. Add Transactions: User will be able to add credit or debit (Len–Den) transactions (*if authenticated), which gets added in the database so that no data loss occurs. User is able to view their transactions once its added and the net amount is readjusted.

3. Modify Transactions: User will be able to edit or delete credit or debit (Len–Den) transactions (*if authenticated), which he has already added in case if the user entered wrong amount or want to remove the transaction as a whole. User is able to view their transactions once its updated/modified and the net amount is readjusted.

4. Modify User Profiles: User is able to set their profile picture, name (as shown to others) or user id as per their wish.

5. Ease of Access: The User Interface is user-friendly such that each user finds the functionality easily accessible.

6. Notify Connections: User can notify connection through verified* e-mail about the dues and payments.

7. View Summary: User can view summary of each conversation to remove hassle of calculating the net amounts and also view the transaction in order they.ve been added.

8. Group Splitting: User can in addition to p2p conversations, make groups to handle amounts which need to be settled among more than 2 people and the bill is split among the rest of the members

9. Add Connections: User can add connections through searching their user id.

Non-Functional Requirements

User Interface Requirements

- A simple and responsive system.
- Web app, consistent in all interfacing screens or devices
- Details of any user and conversations will be activated in the displayed mode and in the database quickly.
- Flexible navigation to and from displayed panels or pages.

Usability Requirements

Every user will have its own type of interface, with the attributes and actions they can perform there. It will be designed in a way that allows modifications to be made, as it has to be updated time after time to fulfil transactions requirements and to manage possible error occurrences.

Performance

The mobile and web application will be an online platform with multiple users. It uses multi-tier architecture which consists of: user client, middle tier, and application server.
Performance is dependent on:
1. Written Code - in order to make the performance of the software better we have insured that we use the optimal code practices etc.

2. Responsive time - ***Example:*** Search results for conversations and other details should be populated within an acceptable time limit.

3. Speed of internet connection

Maintenance

System is maintainable and useable; it is made in the form that later on if required it can be improved by adding more functionalities. This software uses Flutter, Dart VM for mobile client app and React JS for web client app and Python-Flask for the Application Backend Server. The system will be updated continuously with different features and extra features based on user reviews.

Portability

The mobile application is accessible to android users. Web application can be accessed in any platform and browser as far as they have internet connection. In the near future the iOS version might be developed.

Security

Our web and mobile application are connected to same backend. So, security of the application is taken care of by the backend in the following ways:

- Ensures that users and client applications are identified and that their identities are properly verified.
- Ensures that users and client applications can only access data and services for which they have been properly authorized.
- Ensure that parties to interactions with the application or component cannot later repudiate those interactions through JWT verification.
- Ensure that confidential communications and data are kept private.
- Detects attempted intrusions by unauthorized persons and client applications.
- Ensures that unauthorized malicious programs do not infect the application or component.
- Ensure that system maintenance does not unintentionally disrupt the security mechanisms of the application, component, or center.

Authorization and Authentication

Authorization and Authentication part is done by using JWT tokens and some of the flask web framework security tools. User's authorization to application is guaranteed by password and username and a verified email address. Without email verification they cannot have full access to the features of application.

On the other hand, User Authentication is done by using JWT, JSON web token (JWT), pronounced "jot", is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Again, JWT is a standard, meaning that all JWTs are tokens, but not all tokens are JWTs.

Because of its relatively small size, a JWT can be sent through a URL, through a POST parameter, or inside an HTTP header, and it is transmitted quickly. A JWT contains all the required information about an entity to avoid querying a database more than once. The recipient of a JWT also does not need to call a server to validate the token.

Password Encoding

Passwords of the users are hashed with SHA-256 encoding scheme. A random salt is used while generating hash and when user is to be verified,

Then backend checks the hash of the user entered password against the salt used while signing up to ensure authenticity of the user signing in.

Model of the Project: Waterfall Model

The Waterfall model is that the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in an exceedingly linear sequential flow. This suggests that any introduce the event process begins as long as the previous phase is complete. during this waterfall model, the phases don't overlap.

All these phases are cascaded to every other within which progress is seen as flowing steadily downwards (like a waterfall) through the phases. the following phase is started only after the defined set of goals are achieved

for the previous phase and it's signed off, that the name "Waterfall Model". during this model, phases don't overlap.

The major reason why we elect to travel ahead with this model was because it allows for apportionment and control. By following this model, we could schedule all our work beforehand and that we were able to set deadlines for every stage of development and also the product proceeded through the event process model phases one by one.

The project scope stays relatively static, meaning cost and timelines can be determined early on in the project. By completing a full design early in the project, changes to systems stay minimal, meaning the cost to fix and alter designs is kept low. A structured approach to a project means that everyone understands what needs to be done and when. SMEs can effectively plan their time over the fixed period. By having detailed documentation and designs, a project can lose key members without too much hassle since the documentation describes in reasonable detail how any SME of the product or skill are needed to complete the work.

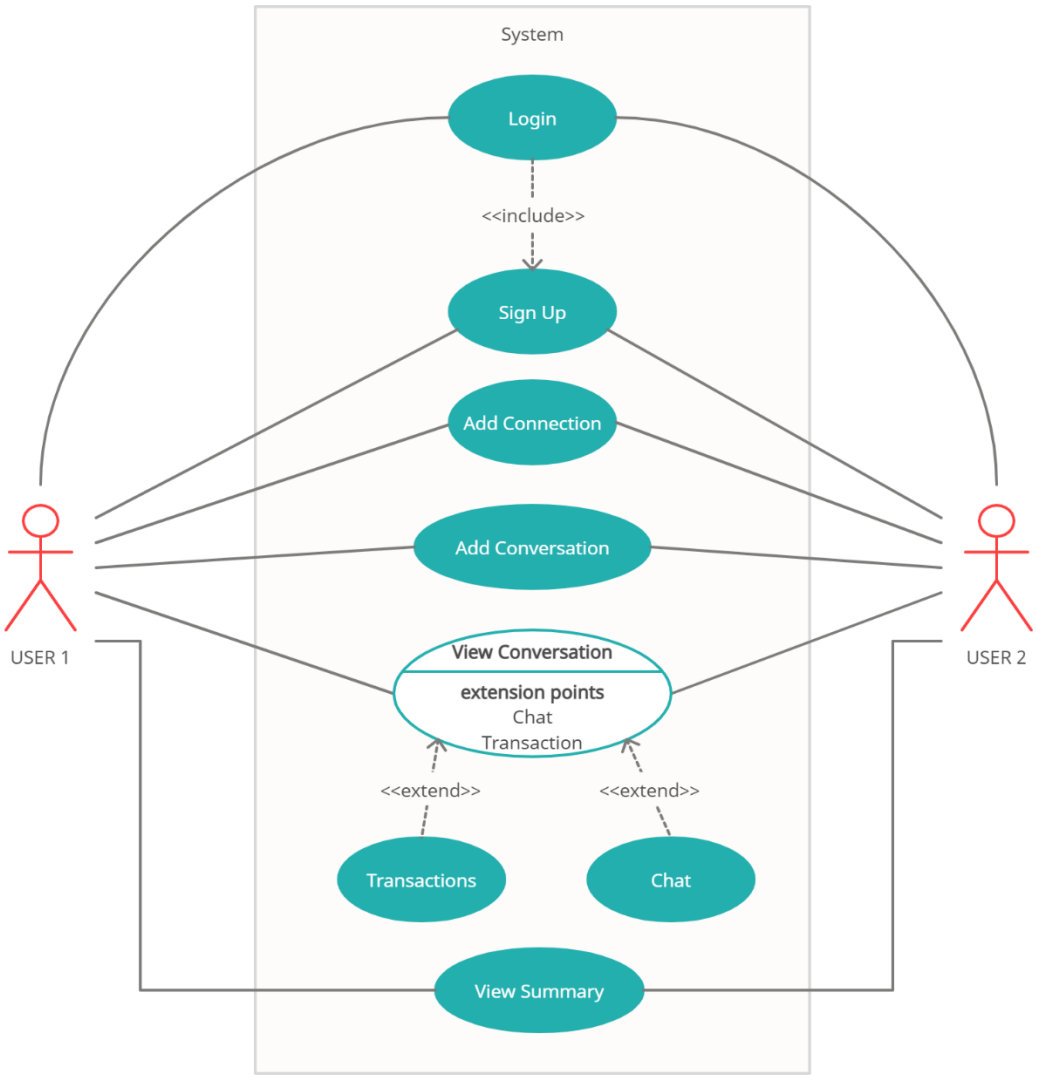
While some may argue this is a burden rather than a benefit, the fact remains that the waterfall model forces the project, and even the organization building said project, to be extraordinarily disciplined in its design and structure. Most sizable projects will, by necessity, include detailed procedures to manage every aspect of the project, from design and development to testing and implementation.

Other Benefits of Waterfall model:

- Planning resources for Waterfall is generally easier as you know exactly when everything will start and end
- Ideal for small projects, where the speed of delivery is not of utmost importance
- Situations where similar projects will be carried out in the future, allowing the project plan to be reused, making use of the heavy documentation that was drawn up previously.

System Designing

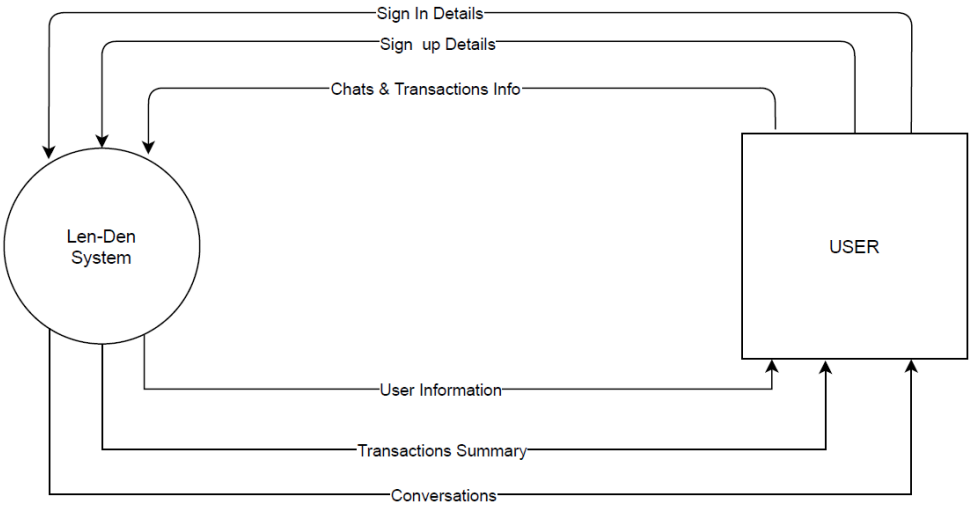
Use Case Diagram



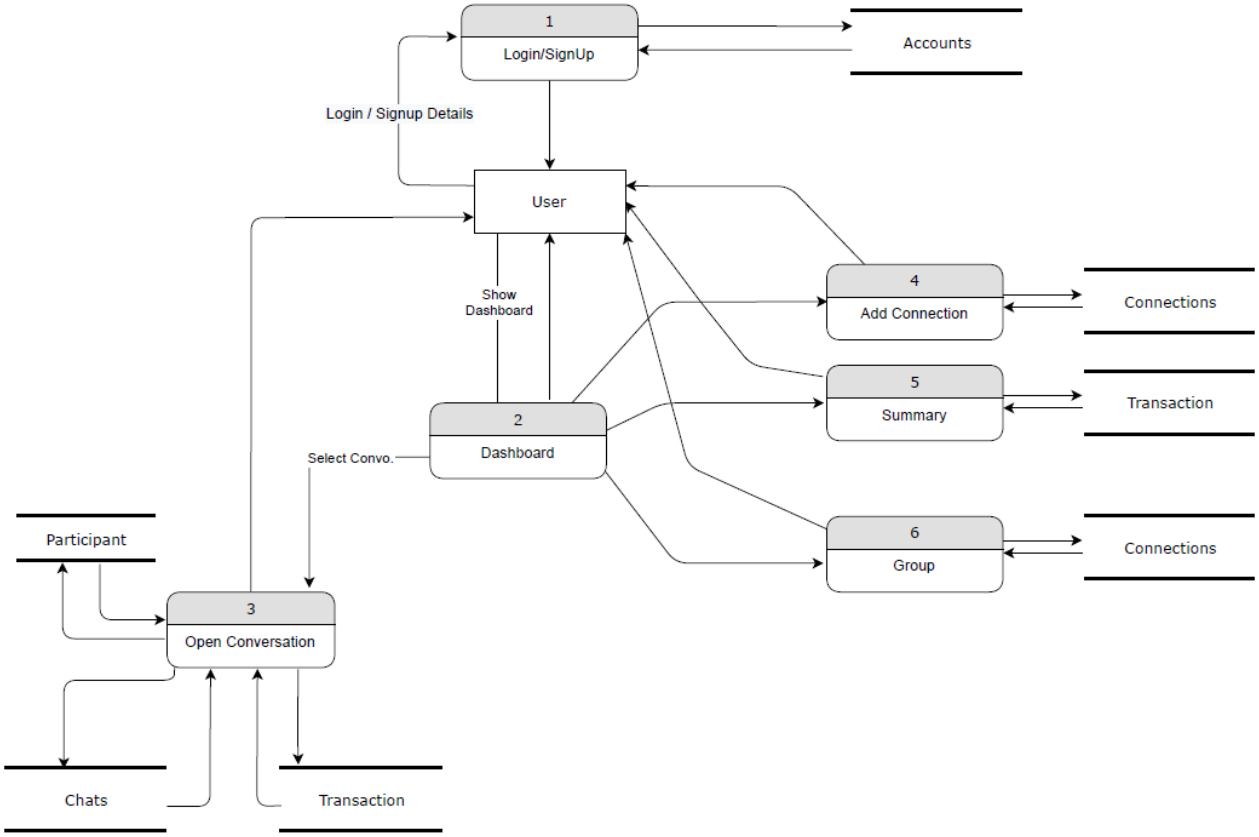
- It only summarizes some of the relationships between use two users.
- It does not show the order in which steps are performed to achieve the goals of each use case.

Data Flow Diagrams:

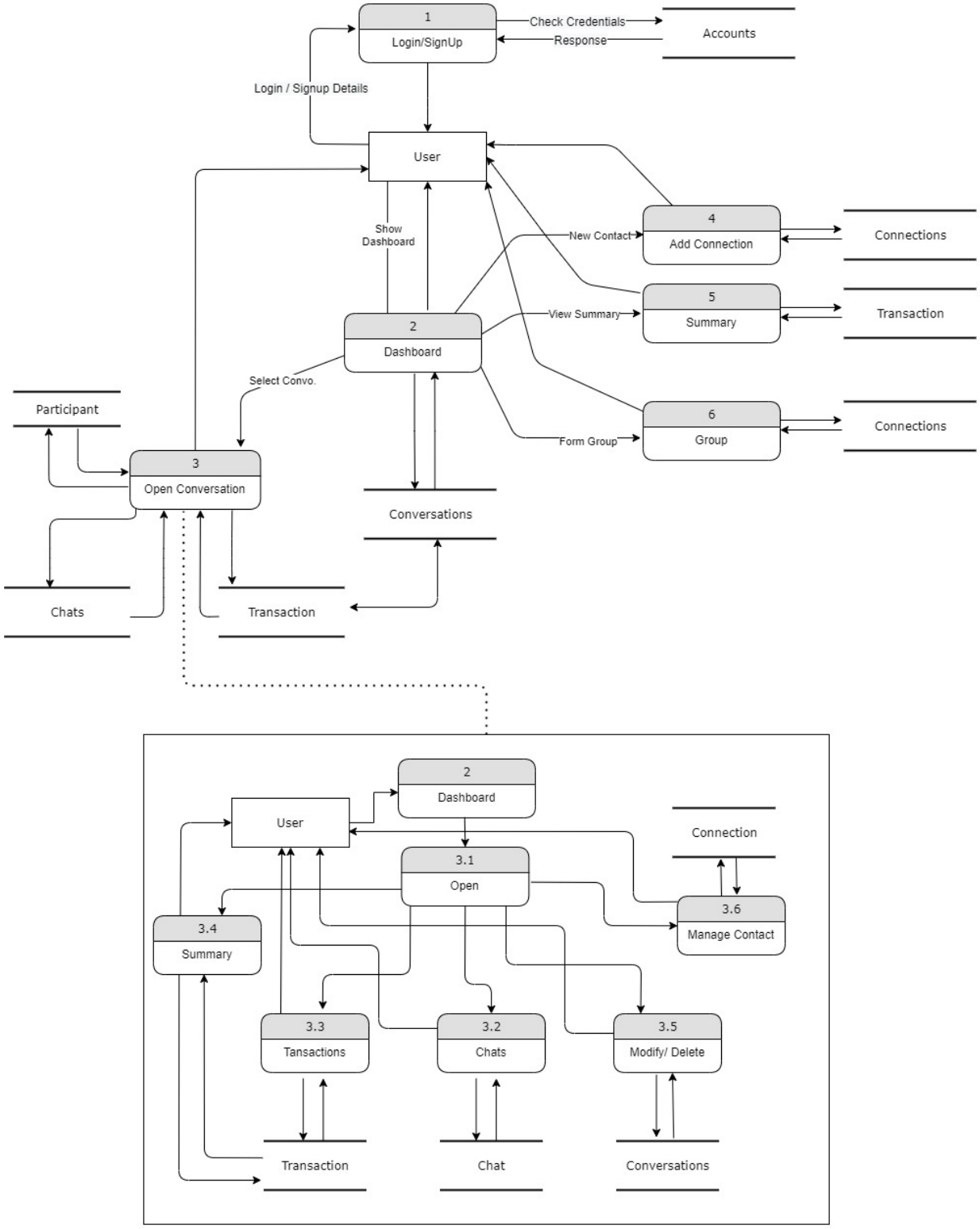
Level 0



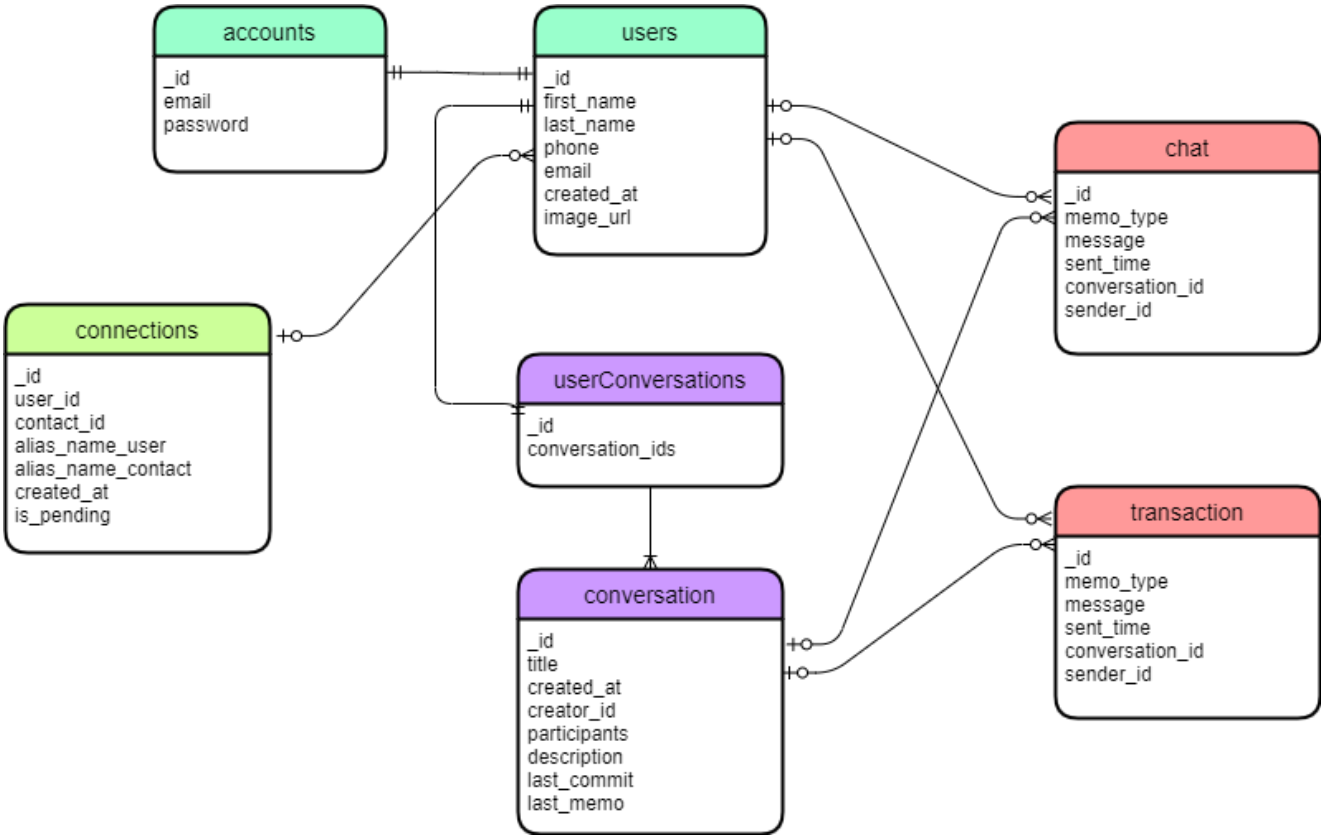
Level 1



Level 2


1
Login/SignUp
Check Credentials
Response
Accounts
Login / Signup Details
User
Show
Dashboard
4
Add Connection
New Contact
Connections
5
Summary
View Summary
Transaction
2
Dashboard
Select Convo.
Participant
6
Group
Form Group
Connections
3
Open Conversation
Conversations
Chats
Transaction
2
Dashboard
User
Connection
3.1
Open
3.6
Manage Contact
3.4
Summary
3.3
Tansactions
3.2
Chats
3.5
Modify/ Delete
Transaction
Chat
Conversations

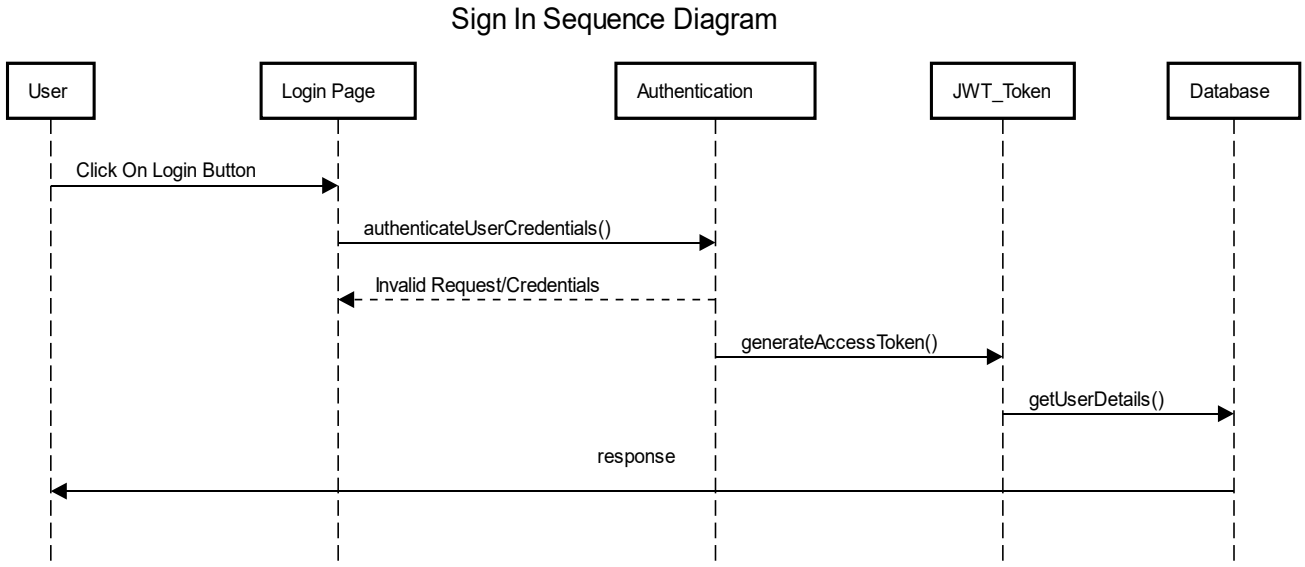
ER Diagram or Relational Schemas



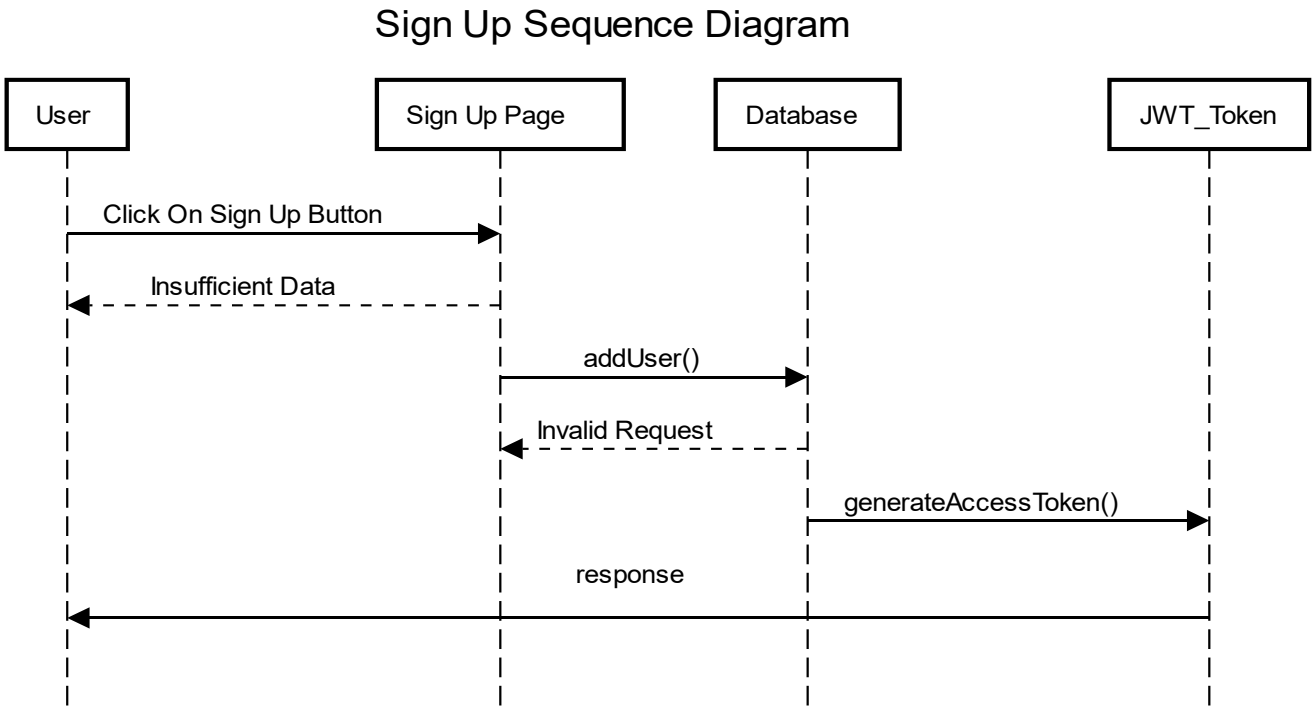
Here,

- users and account schema collect the user credentials and encrypt them before storing into the database.
- userConversations and conversation are used to retain the details of conversation a particular user is into, and show the chats and transaction in that particular conversation.
- connections store the mutual relationship between the users to show that they are connected and can interact with each other.
- chats and transaction collections store every chat or transaction with the relevant conversation id to quickly find the queried chats and transactions.

Sequence Diagrams

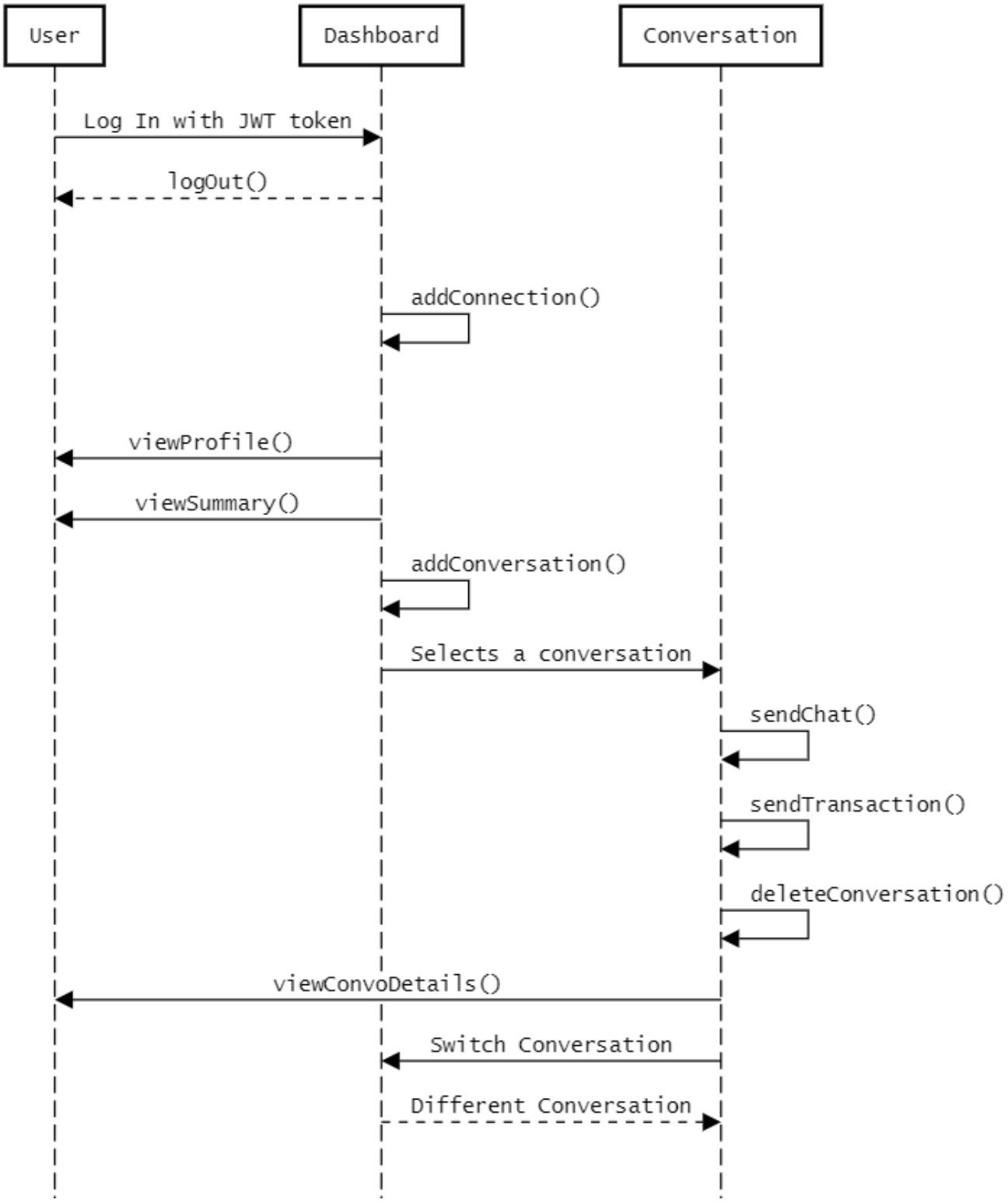


Here, user signs in with correct credentials and the application server responds to the client with corresponding user details and a JWT toke which is further used in every authenticated action which the user can perform.



Here, if a user wants to sign up for the application, can provide sufficient information like first name last name email phone etc. and if the email and user id does not already exist, he/she will be successfully logged in with a new account.

Dashboard Sequence Diagram



Implementation

We propose a system where the user has ease-of-access while using the application.

It will become very convenient to manage your friends' entries, get reminders for the due date, get a complete summary of all the contact's entries in one place with a balance sheet, and become fearless of forgetting or losing information.

Now, following the waterfall model standard approach, we first put our thoughts over the system requirement and the functionalities our product should provide to the user.

The functionalities include:

- **Ease of Access:** Simple and intuitive UI for easy navigations & Manage your ledger account book with a simple interface.
- **Flexible:** The application should be Flexible in adding, cancelling, and deleting a credit or debit entry.
- **Expense & Income:** Gives you a complete summary of your Debits and Credits amounts in one screen.
- **Group Expenses**: Split a particular bill into a group of friends when a specific person has to collect money for the same expense through multiple contacts.

We documented the SRS document of our project which included the initial requirements which modified a bit as we progressed over developing our product based on the constraints.
Now, we come to the 2nd step of the waterfall model, which is system designing. Here we made the UML ER diagram/ Relational Schema followed by the Data Flow Diagrams up to Level 2 and also the Use case diagram.

We used Mongo DB as the database for our project. Mongo DB is a document oriented NO SQL database. The advantage of Document oriented over relation type is the columns can be changed as an when required for each case as opposed to the same column name for all the rows.
Now, we arrive at the implementation step or the coding part of our project, before which we created our database over Mongo Atlas which is a cloud database service, for which we used the `pymongo` package for our backend to access the collections over the cloud database, which is common for both mobile and web application.

pymongo: *The PyMongo distribution contains tools for interacting with MongoDB database from Python. The `bson` package is an implementation of the BSON format for Python. The `pymongo` package is a native Python driver for MongoDB.*

- For the Mobile client application, the runtime environment is Dart VM which uses dart language and flutter is used for mobile UI.
- For the Web client application, ReactJS framework is used which uses the context API to share data that can be considered “global” for a tree of React Components.
- For the common server for both mobile and web client, Flask framework in python is used to serve the APIs to the clients.

Protection

Flask is web framework that provides some security tools for the backend. It offers authenticating HTTP requests using basic HTTP authentication and also provides developers tools to implement their own authentication strategies.

The security components of Flask contain subcomponents which can be used separately to guarantee different security aspects of the application:

Itsdangerous:

- Given a key only you know, you can cryptographically sign your data and hand it over to someone else. When you get the data back you can easily ensure that nobody tampered with it.
- Internally ItsDangerous uses HMAC and SHA-512 for signing by default. The initial implementation was inspired by Django’s signing module. It also supports JSON Web Signatures (JWS).

Passlib:

- Passlib is a password hashing library for Python 2 & 3, which provides cross-platform implementations of over 30 password hashing algorithms, as well as a framework for managing existing password hashes.

Flask / security-http:

- It integrates the core sub-component with the HTTP protocol to handle HTTP requests and responses.

Use of JWT in our application:

Authentication: When a user successfully logs in using their credentials or sign ups using email password, an ID token is returned. According to the OpenID Connect (OIDC) specs, an ID token is always a JWT.
Authorization: Once a user is successfully logged in, client application request to access routes, services, or resources (e.g., APIs) on behalf of that user. To do so, in every request, it must pass an Access Token, which may be in the form of a JWT. Single Sign-on (SSO) widely uses JWT because of the small overhead of the format, and its ability to easily be used across different domains.
Information Exchange: JWTs are a good way of securely transmitting information between parties because they can be signed, so our application can be sure that the senders are who they say they are. Additionally, the structure of a JWT allows us to verify that the content hasn't been tampered with.

Flow of authentication:

When user signs-in or sign-ups, a JWT token is generated containing the expiration time of token (which is 10 days), User-Id, and issuer of the token (which is Authentication-Server) and returned to client side which they can use for further requests to server. The JWT is signed using the private key (which is generated using OpenSSL) with RSA256 asymmetric encryption. Furthermore, when client requests for certain end point, then request is checked against the Authorization header which should contain this JWT token, the token is checked against public key of the authentication server, if token is valid then the request of the client is served according to the Token.

Now, we build the APIs according to our functional requirements and tested the APIs. Simultaneously, we worked over both mobile and web frontends to synchronize the progress with the server part.

Finally, at the testing stage of waterfall model, we used unit testing to test the APIs to test that all the APIs are responding in the correct way for corner cases.

Tech Stack:

- **Database Server**: MongoDB
- **Server Client**: Any web browser / Android
- **Programming Language**: JavaScript, Dart

- **Development tools**: VS Code, Github
- **Design Tools**: Figma

- **SDK (Mobile Application)**: Flutter
- **Environment**: NodeJS, Dart VM

TASK PLANNING & DISTRIBUTION

#	ACTIVITY	Duration
1	**Specify assumptions, constraints and dependencies**	7days
2	**Preparing a detailed SRS (functional, non-functional requirements)**	7days
3	**ERD and DFD**	7days
4	**Scenarios and Use case diagram**	14days
5	**Database**	5days
6	**Code**	21days
7	**Testing**	5days
8	**Deployment**	Ongoing

Testing

There are many test runners available for Python. The one built into the Python standard library is called *unittest*.

unittest has been built into the Python standard library since version 2.1. *unittest* contains both a testing framework and a test runner.

It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework

auth_unit_test.py

```python
import unittest
import json

from auth_server import app
from auth_db import DataBase

BASE_URL = 'http://127.0.0.1:5001/'

BASE_USER = {
    "user_id": "testuser",
    "password": "testuser",
    "email": "test@user.com",
    "phone": "9876543210",
    "first_name": "test",
    "last_name": "user",
}
```

Each test unit must be fully independent. Each test must be able to run alone, and also within the test suite, regardless of the order that they are called. The implication of this rule is that each test must be loaded with a fresh dataset and may have to do some cleanup afterwards. This is usually handled by setUp() and tearDown() methods shown below.

```python
class TestFlaskApi(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.db = DataBase
        self.app.testing = True
```

setup() is run before each unit test.

```python
    def test_successful_signin(self):
        payload = json.dumps(BASE_USER)
        response = self.app.post(
            '/signup', headers={"Content-Type": "application/json"}, data=payload)

        payload = json.dumps({
            "user_id": "testuser",
            "password": "testuser"
        })

        response = self.app.post(
            '/signin', headers={"Content-Type": "application/json"}, data=payload)

        data = json.loads(response.get_data())
        self.assertEqual('JWT', data['token_type'])
        self.assertEqual(200, response.status_code)
```

A random test case is chosen.

```python
    def tearDown(self):
        # Delete Database collections after the test is complete
        for collection in self.db.list_collection_names():
            self.db[collection].delete_many({})


if __name__ == "__main__":
    unittest.main()
```

Clear the databse before running next unit test.

These are the Test results for unit tests of Authentication and Application Server.

```
.......
----------------------------------------------------------------------
Ran 7 tests in 6.230s

OK

(.lenden) C:\Users\DELL\Desktop\Software\Lenden\server>
```

auth_unit_test

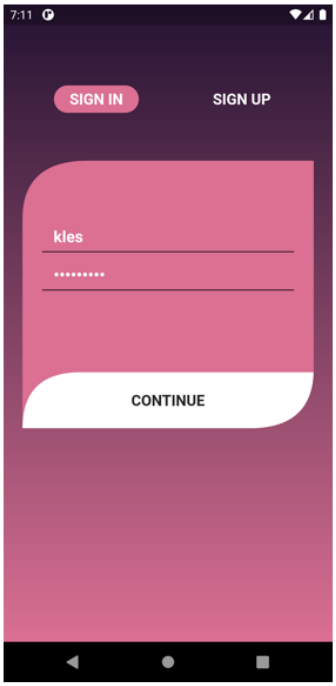
```
(.lenden) C:\Users\DELL\Desktop\Software\Lenden\server>python -W ignore app_unit_te
st.py
.....................
----------------------------------------------------------------------
Ran 21 tests in 18.114s

OK

(.lenden) C:\Users\DELL\Desktop\Software\Lenden\server>
```

app_unit_test

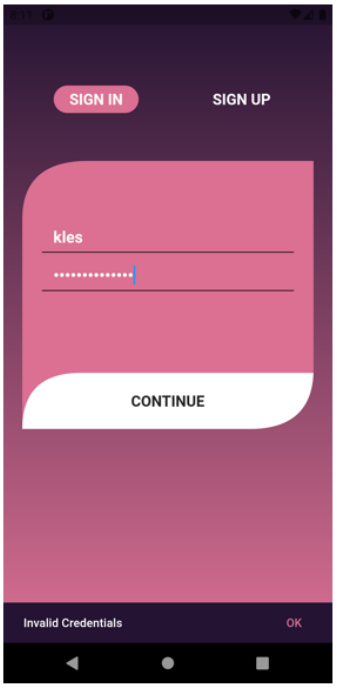
APPENDIX -> Screenshots

- Mobile-client

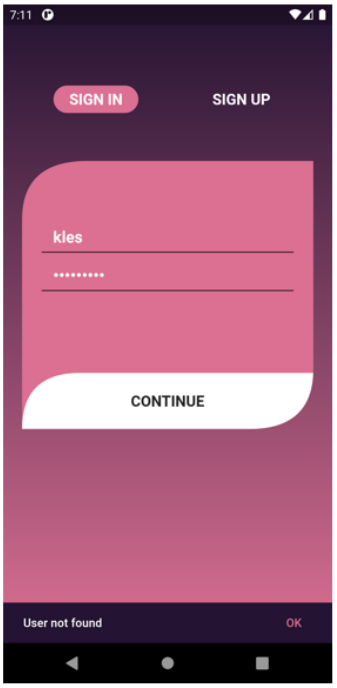
Login Page



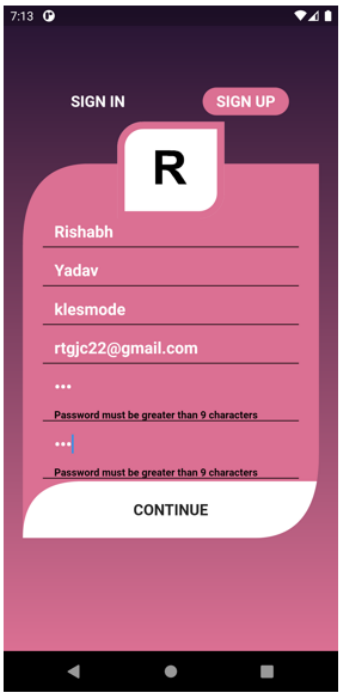
Invalid Credential

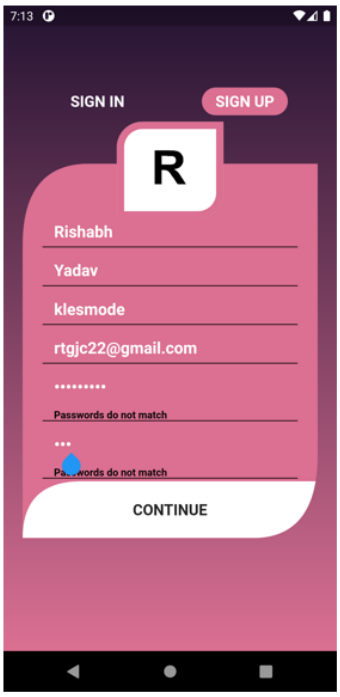


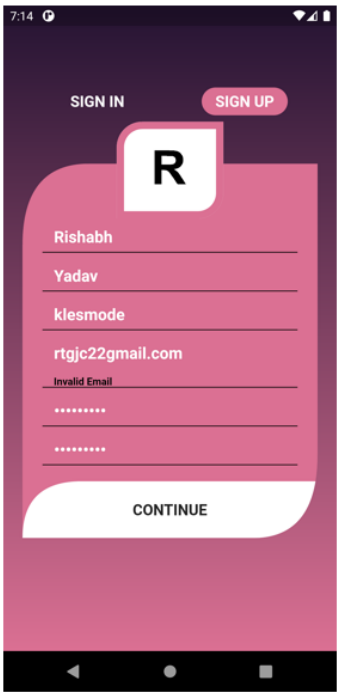
User not found



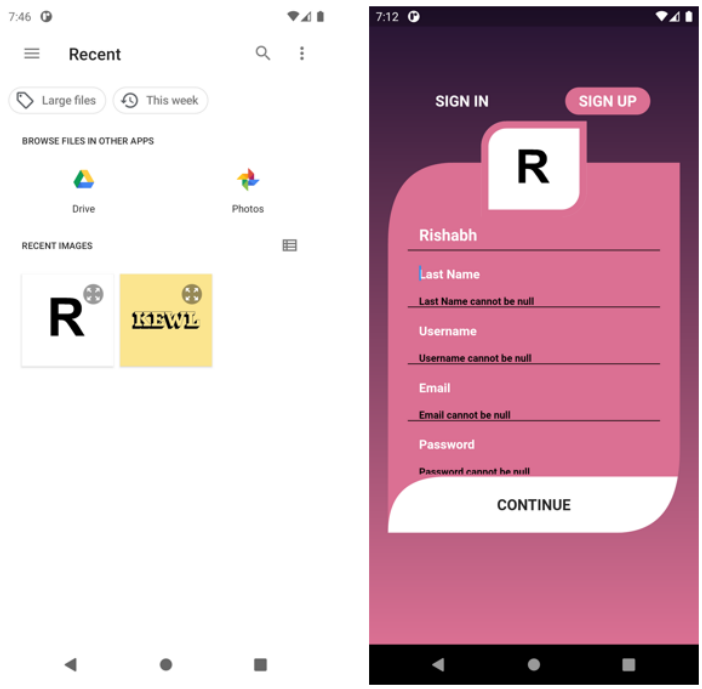
Registration Form: Form-control on client side



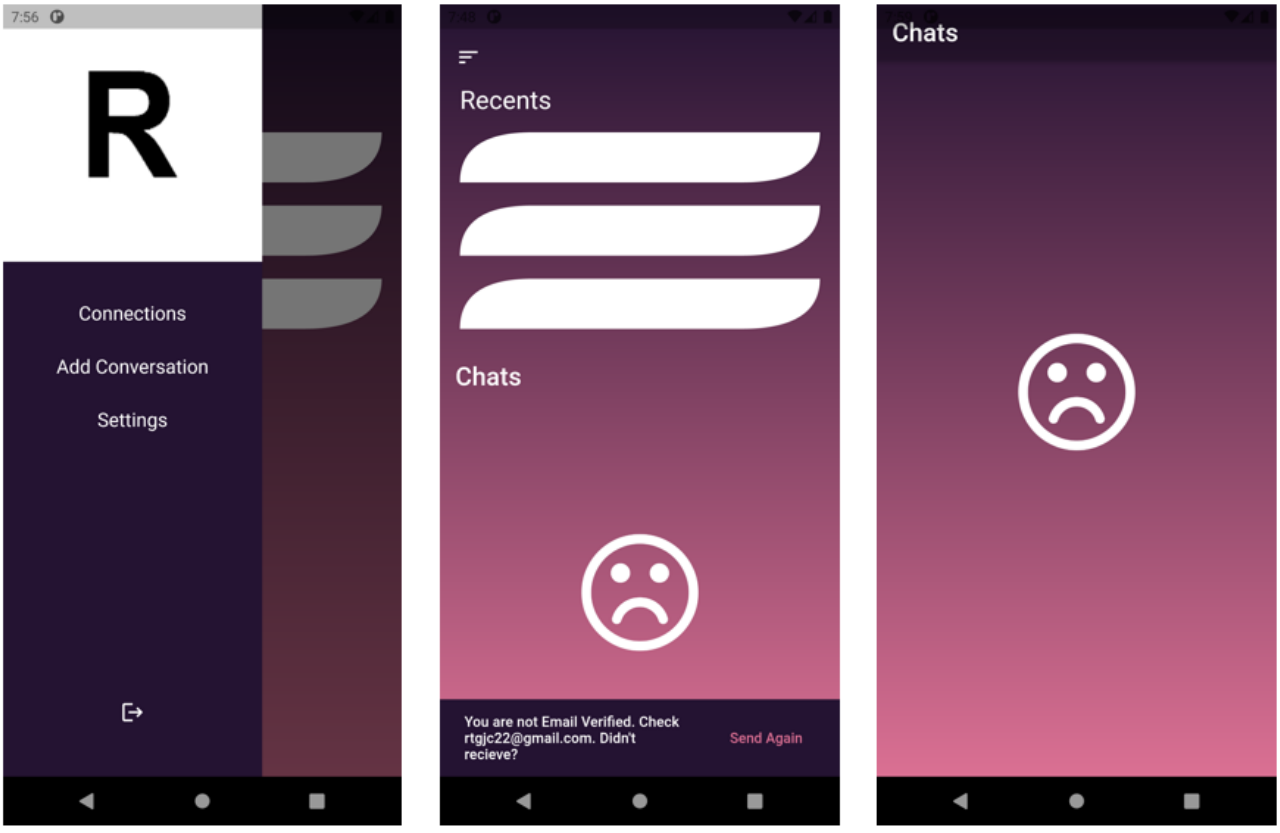




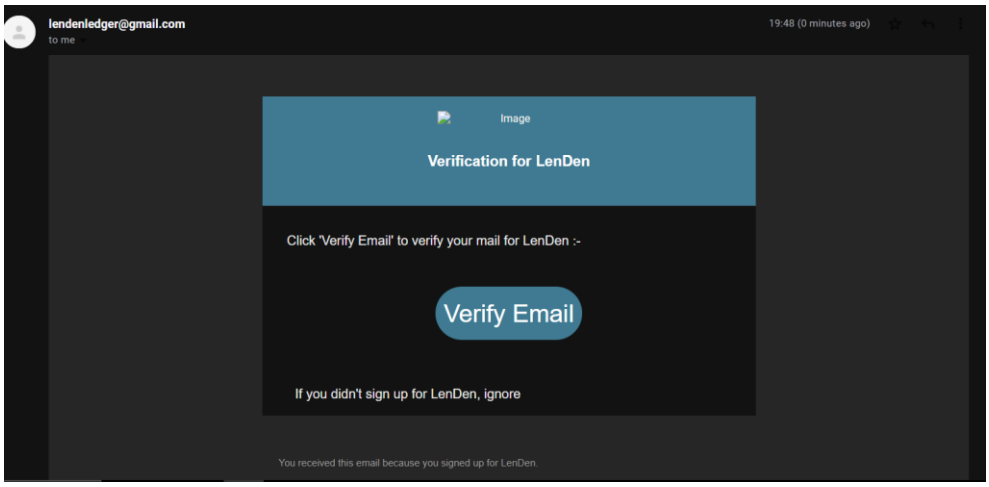
Profile Picture Selection



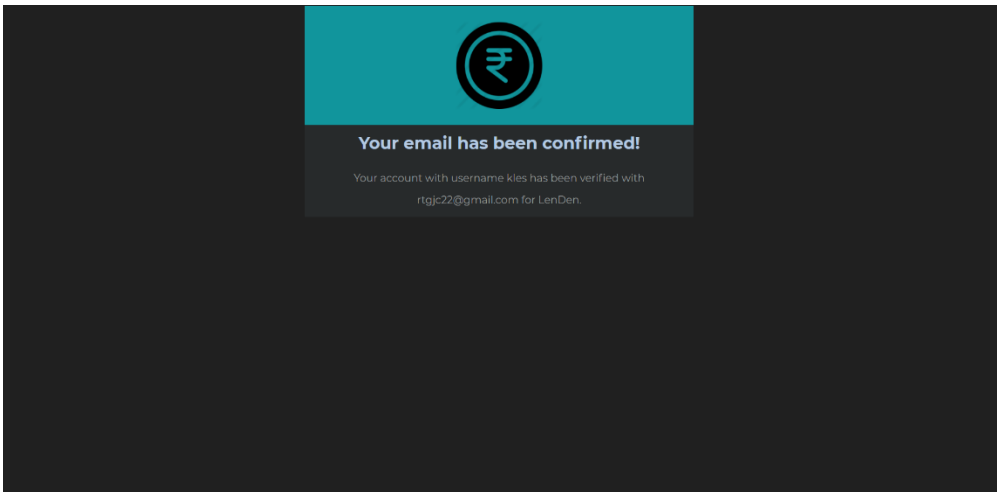
On successful signup



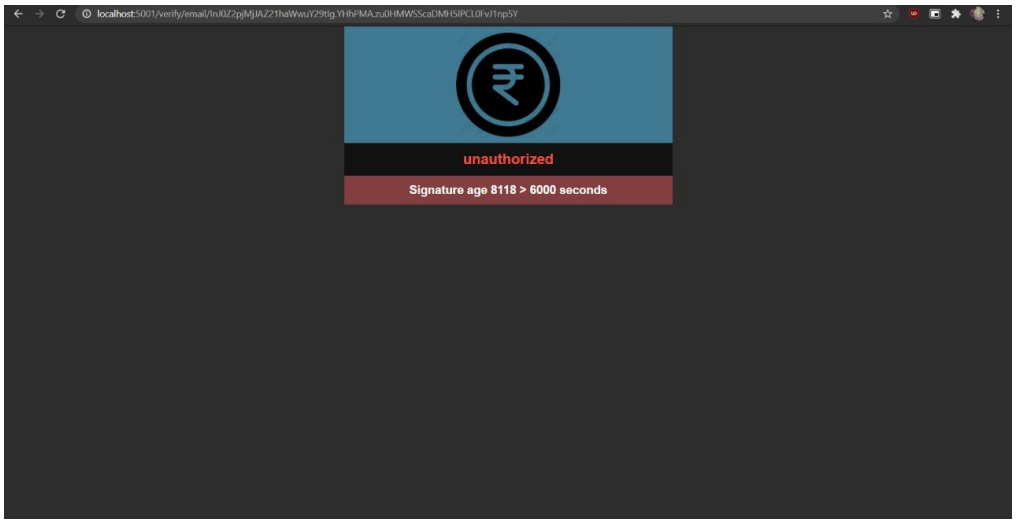
Upon clicking the email received in the corresponding email



On verifying the email

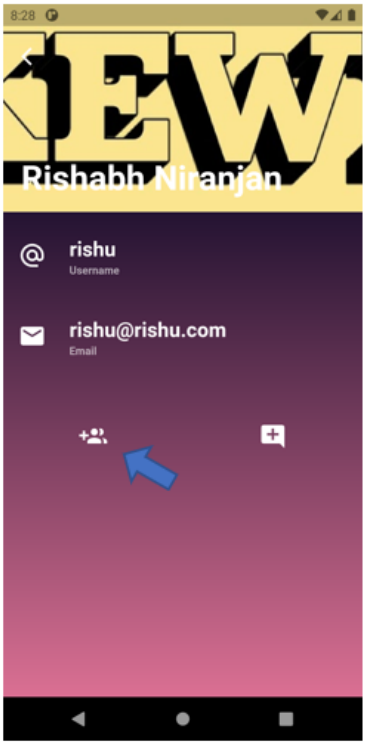


If email verification token expires

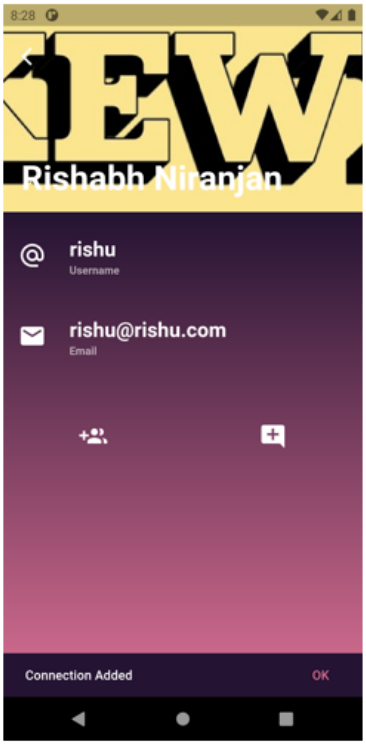


Search User

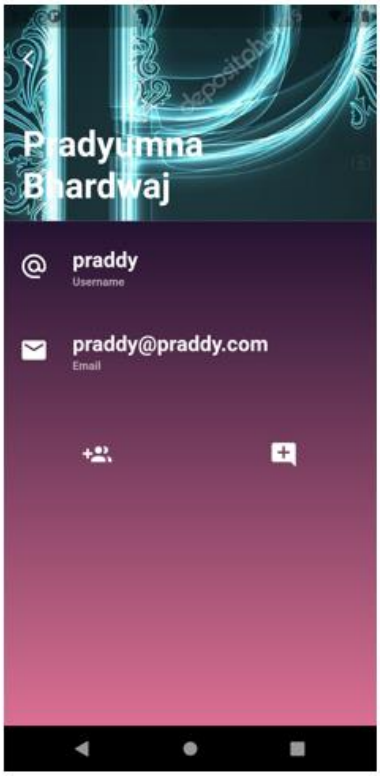
Open User Profile



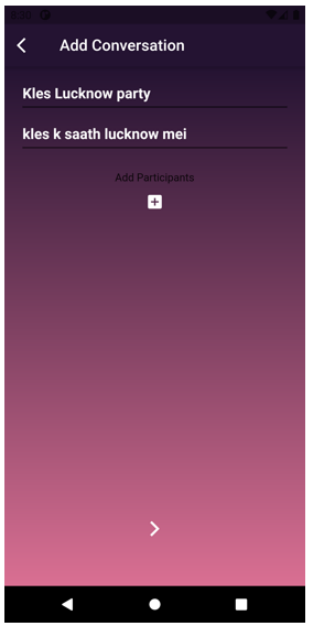
Make Connection



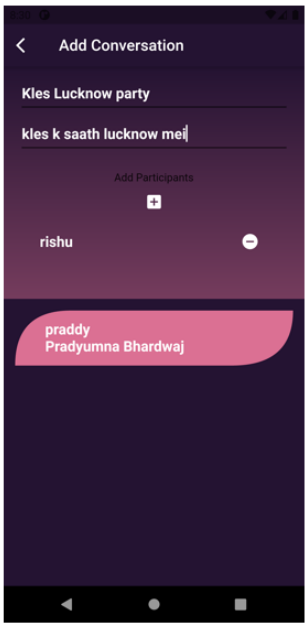
User can add more connection and view all his/her connections at one place



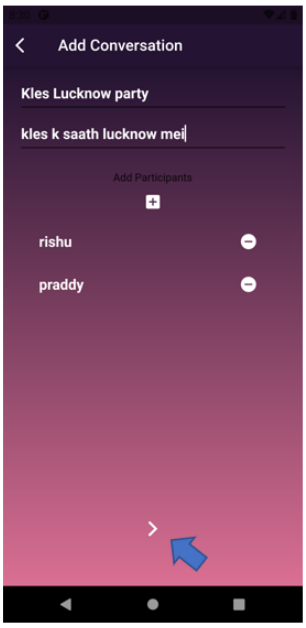
Adds a conversation



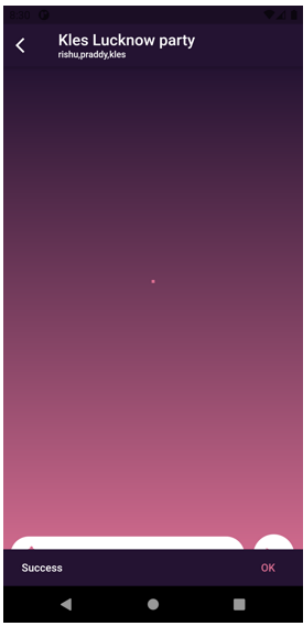
Selects the participants



Creates Conversation

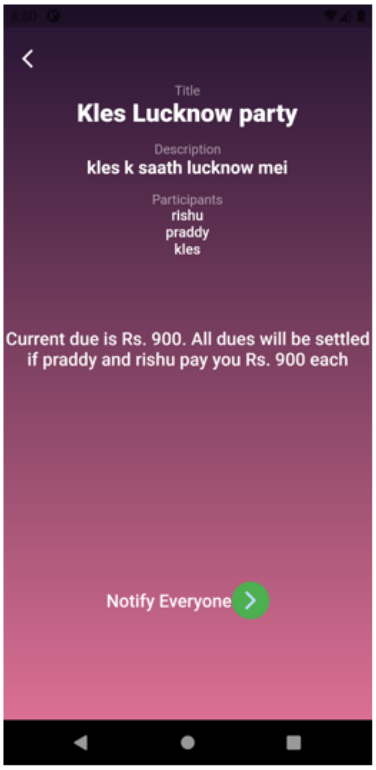


New Conversation is added

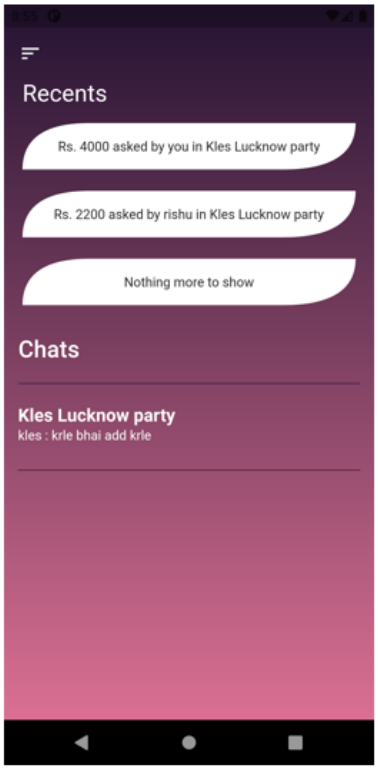


Chat and Transaction in a conversation

View summary



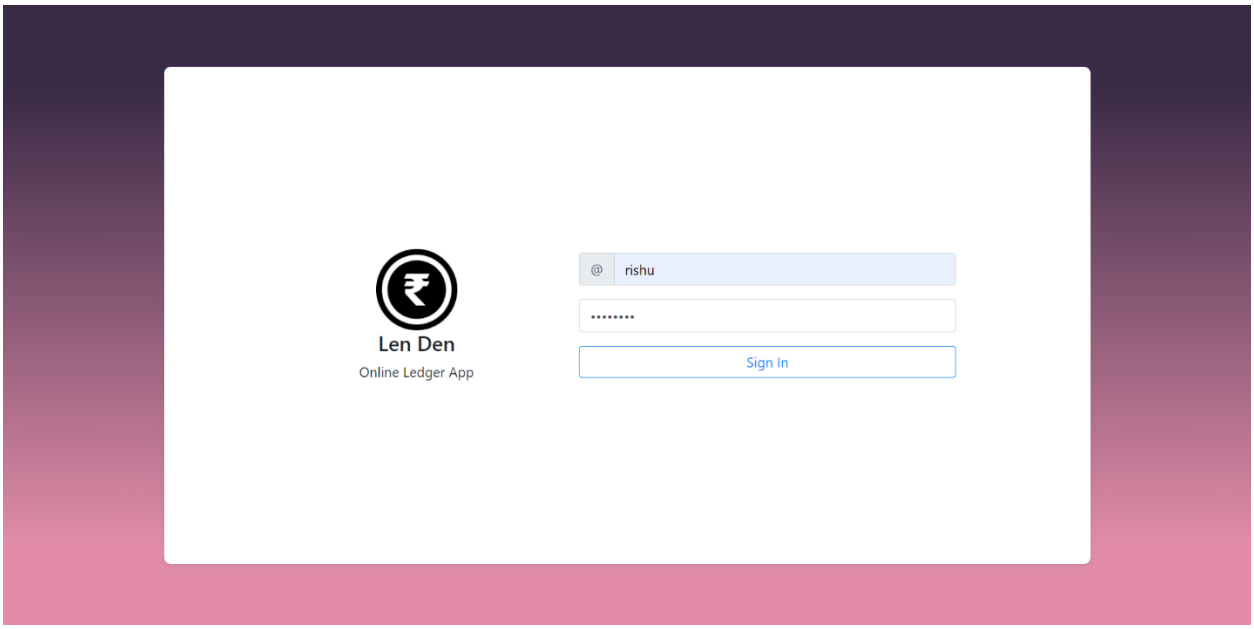
View Conversations



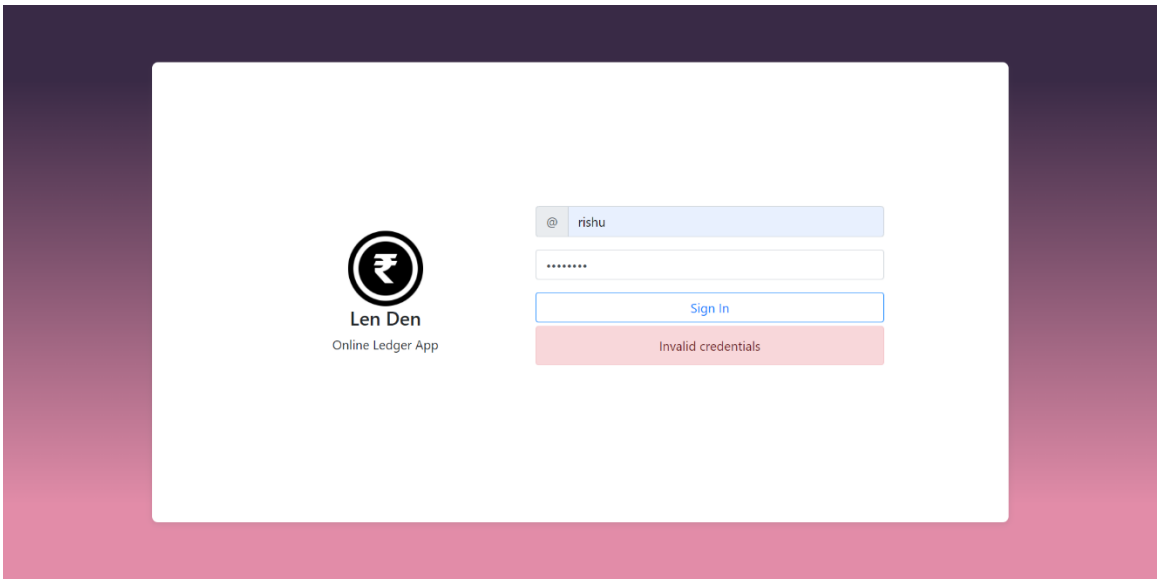
Add a new transaction

- Web-client

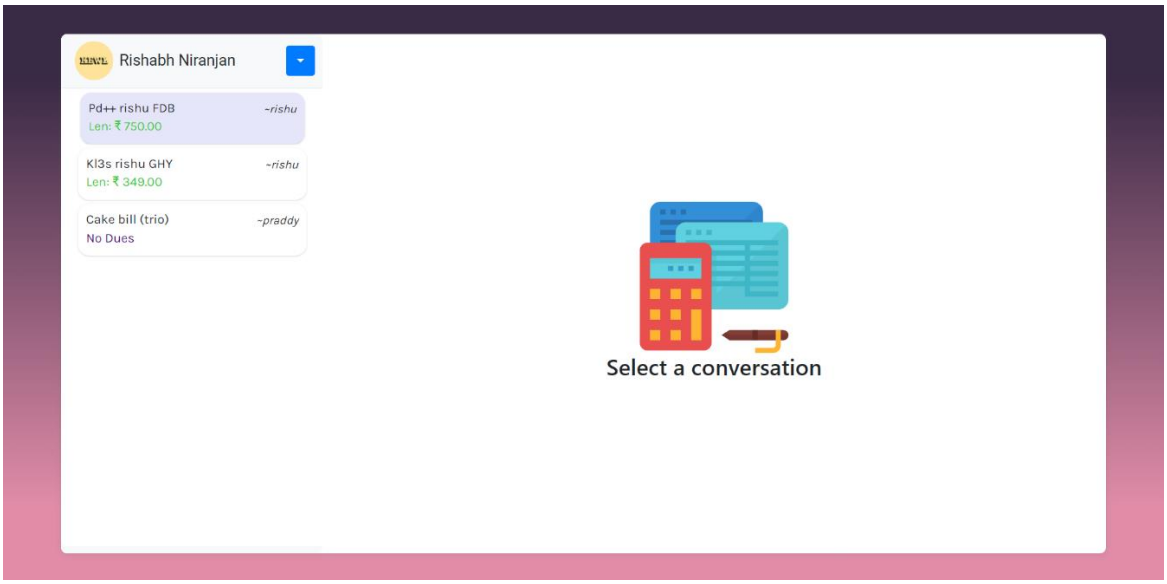
Sign in Window

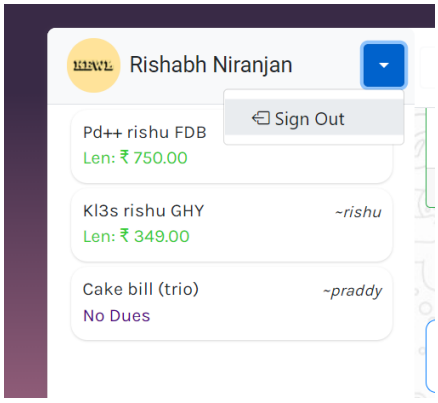


Invalid Credentials



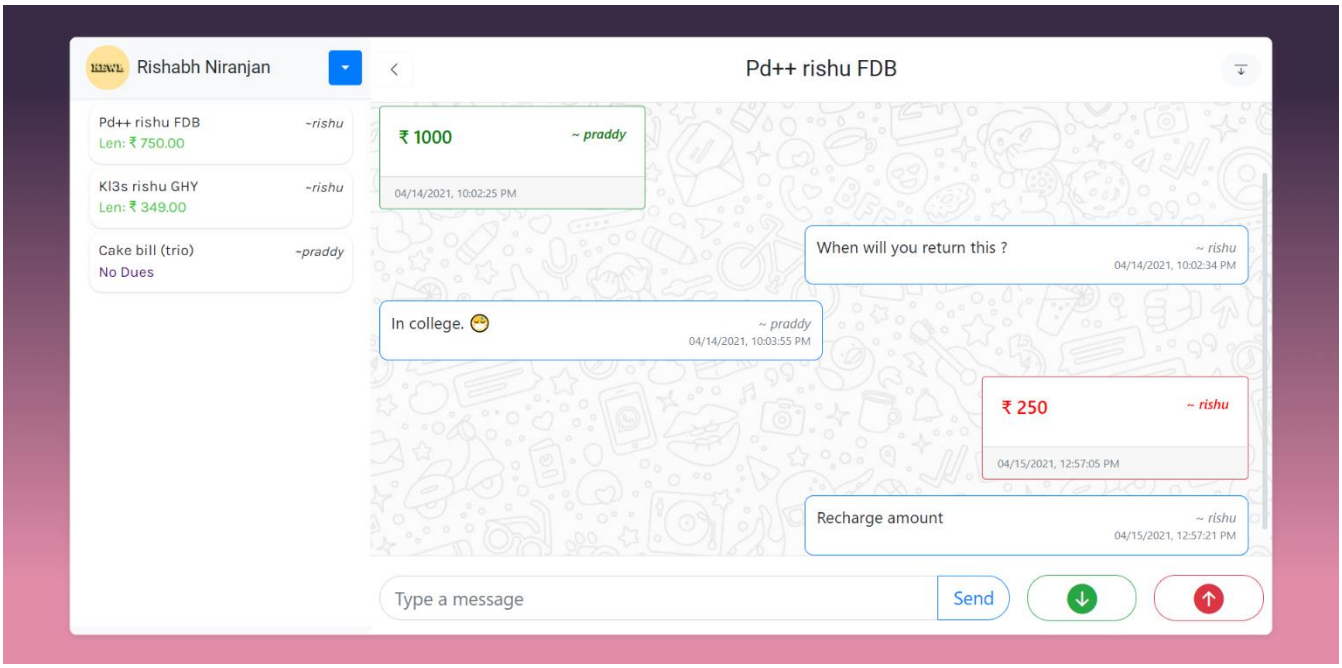
Upon successful sign in, Dashboard



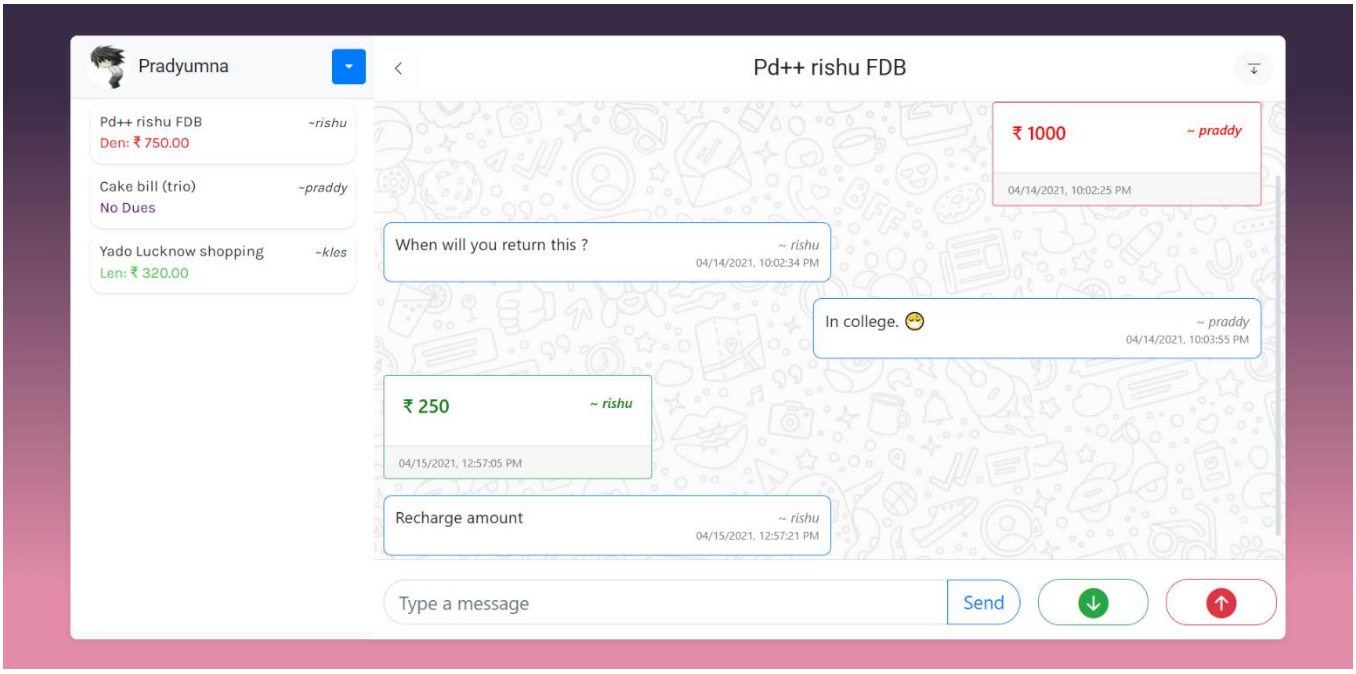


Sign Out

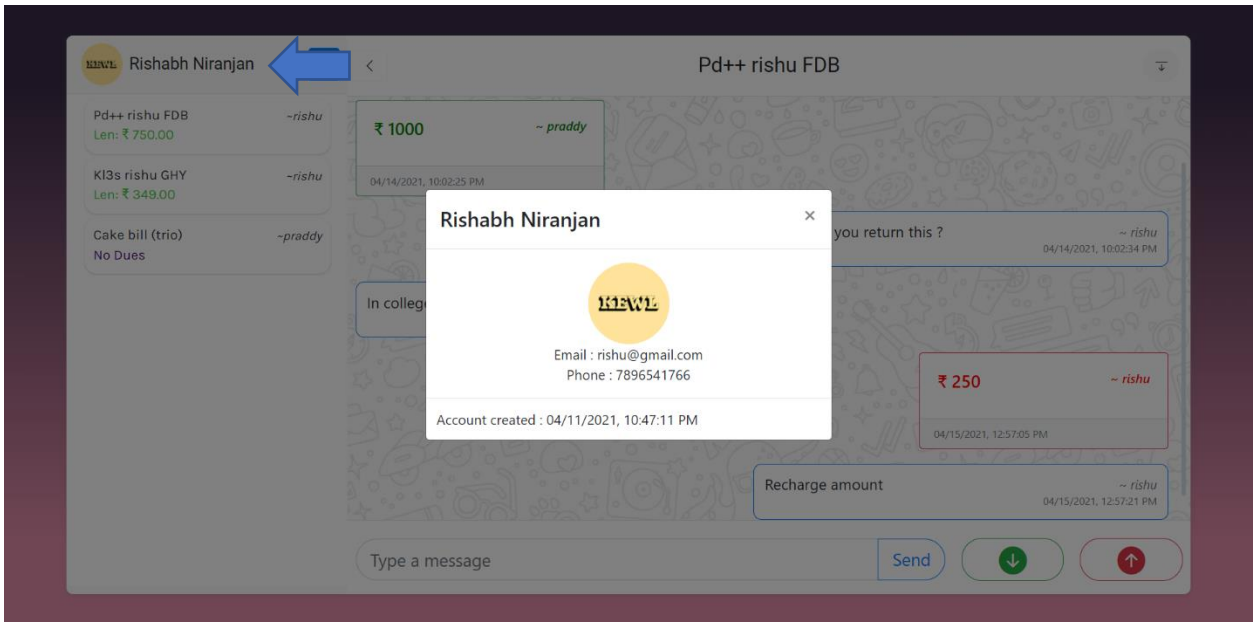
Particular conversation w.r.t. user 1



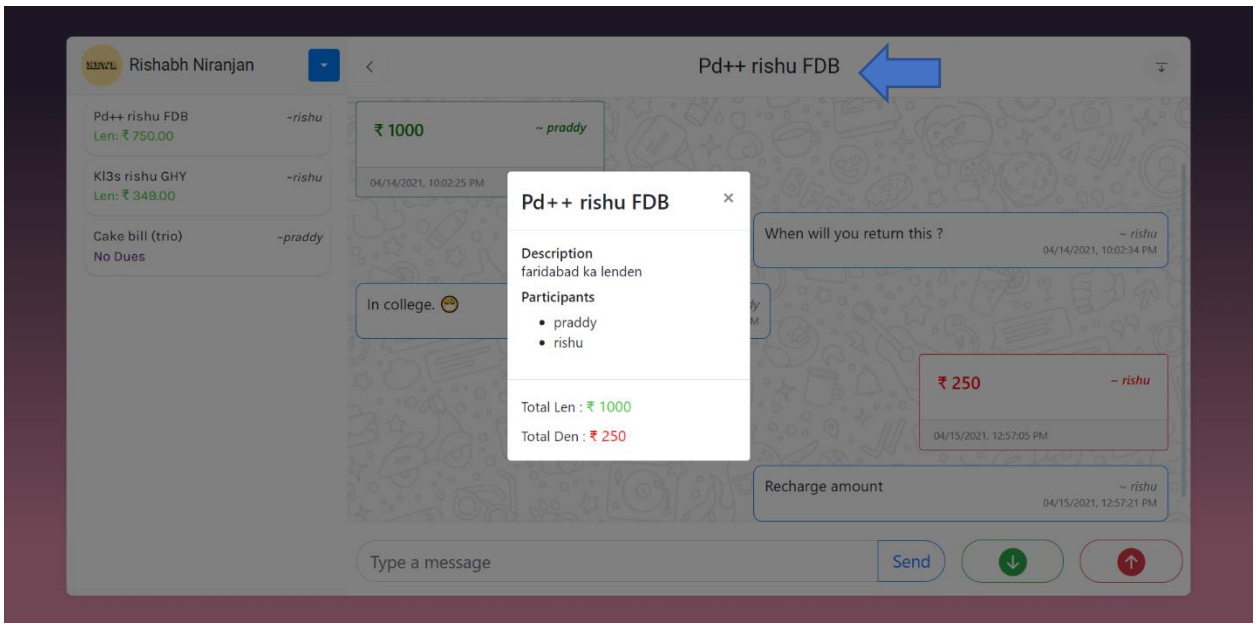
Particular conversation w.r.t. user 2



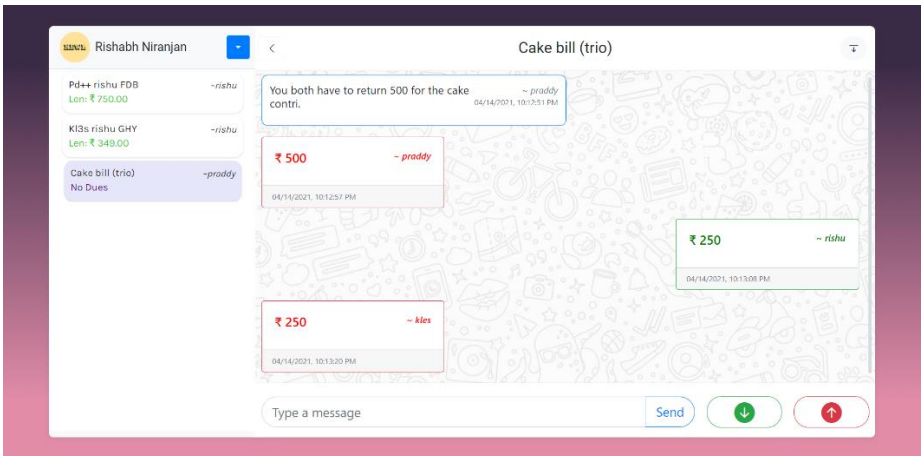
User Profile Information

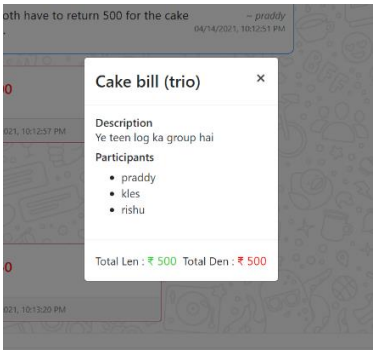


Conversation Information

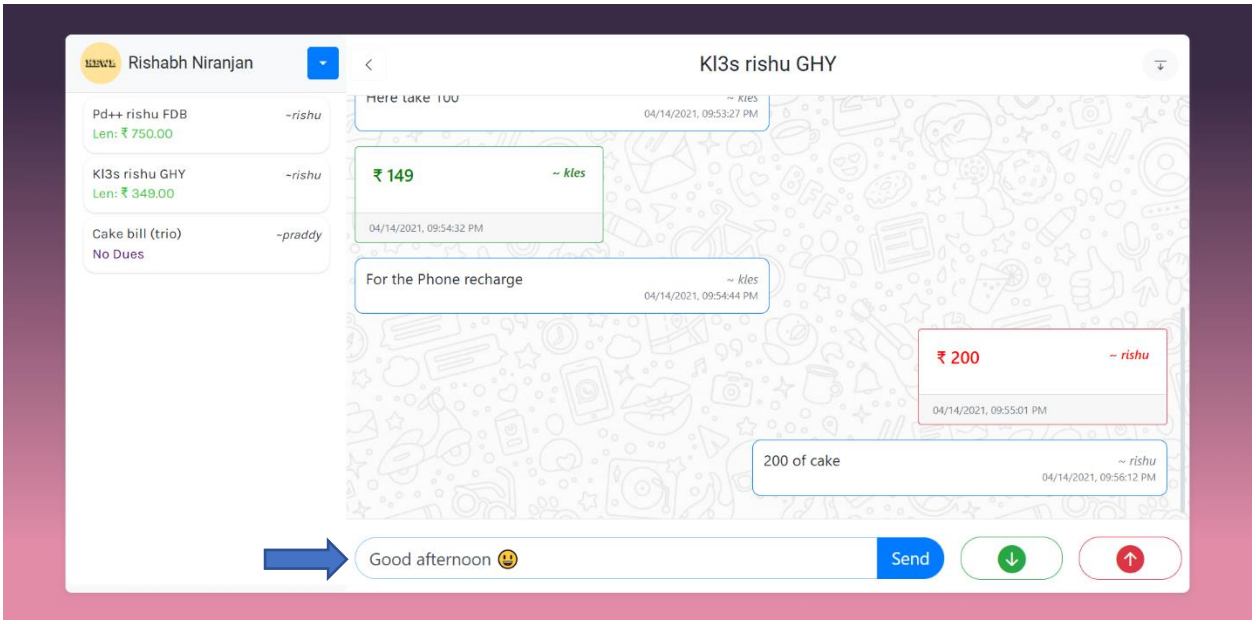


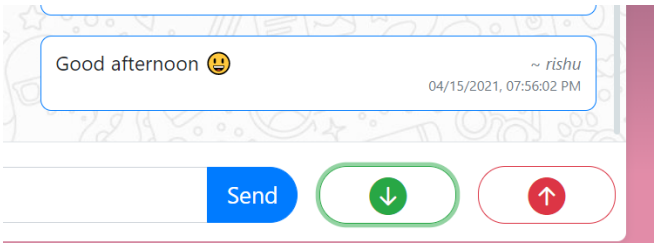
Group for more than 2 users to facilitate bill splitting





Sending a text message:





Transaction Flow